State Machines and Regular Languages

October 31, 2012

CS 81: Computability and Logic

Computability

- ✓ How can I prove that a computer can solve some specific problem?
  - E.g., sorting or the Traveling Salesperson Problem
- ✓ How can I prove that **no** computer can solve a specific problem?
  - If I prove it today, will it still be true tomorrow?

Describing the Problems to be Solved

Assumption 1: Inputs are *finite strings from some finite alphabet of characters.*

- ✓ `"37"`
- ✓ `"3*7=21"`
- ✓ `"sort([3,1,4,1,2], [1,2,3,4])."`

Assumption 2: We care about *decision problems* (i.e., answer is just yes or no)

- ✓ Is 37 prime?
- ✓ Does $3 \times 7 = 21$?
- ✓ Is $[1, 2, 3, 4]$ the result of sorting $[3, 1, 4, 1, 2]$?
- ✓ ⋮

Combining the Assumptions

Any *decision problem* is *equivalent to asking*

"Is the input a member of the set L?"

for a suitable set L.

- ✓ For primality testing, $\mathrm{L} = \{2, 3, 5, 7, 11, 13, 17, 19, \ldots\}$.
- ✓ For multiplication-correctness,

$$\mathrm{L} = \{ \begin{array}{llll} 0\times0=0, & 0\times1=0, & \ldots \\ 1\times0=0, & 1\times1=1, & \ldots \\ 2\times0=0, & 2\times1=2, & \ldots \\ \vdots & & \end{array} \}$$

Conclusion

A one-to-one correspondance:

(computational) problems to solve

sets of finite strings.

These sets are called “formal languages”

Idealized Computers

We need a precise (mathematical) definition, abstracting away details that might change.

- ✓ Operating System
- ✓ Processor speed
- ✓ Memory capacity
- ✓ Power source (electricity, natural gas, dilithium, . . .)
- ✓ Construction materials (silicon, graphene, legos, . . .)
- ✓ Programming language (Java, Racket, Prolog, HMMM, . . .)
- ✓ Architecture (single core, multicore, manycore, GPU, VLIW, . . .)
- ✓ Data representation (ASCII, Unicode, binary, trinary, . . .)

Idealized Computers

State Machine, which

- ✓ A set of possible "configurations" (states)
- ✓ Rules for how the system proceeds from one state to another
  - ▶ Depends on current state *and* current input
- ✓ Accept or reject, based on the input this far.

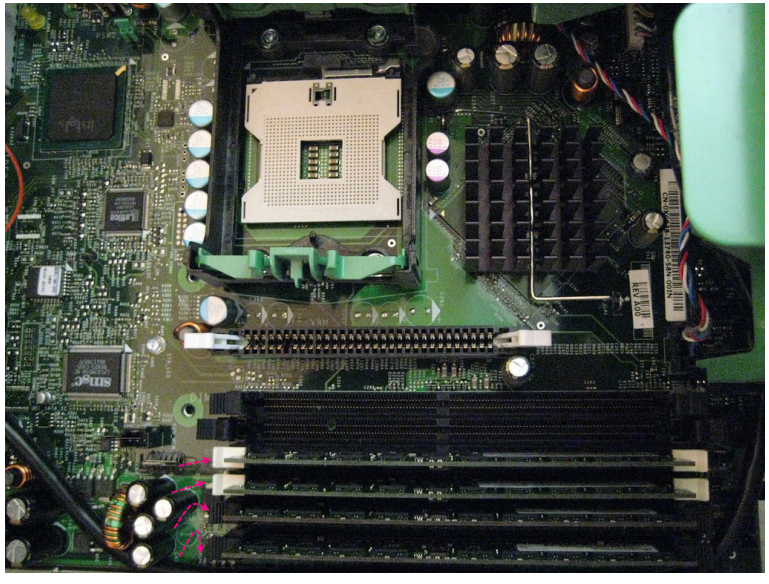
What is a state?

The *state* of a system at some instant consists of the all internal information needed to figure out how to proceed.

State?

http://www.flickr.com/photos/alexbrn/5035170693

State?

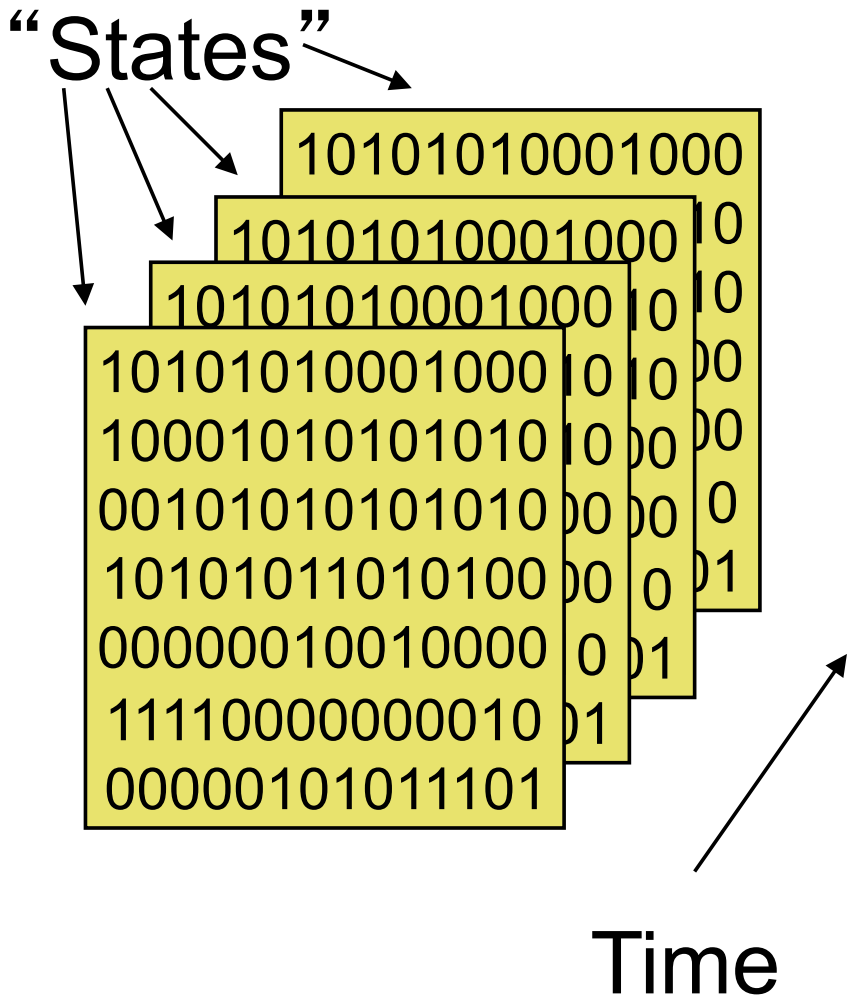


http://www.flickr.com/photos/cambodia4kidsorg/3019948738

Many systems have more than 50 states!

State Machines are everywhere!

Computers: State changes over time



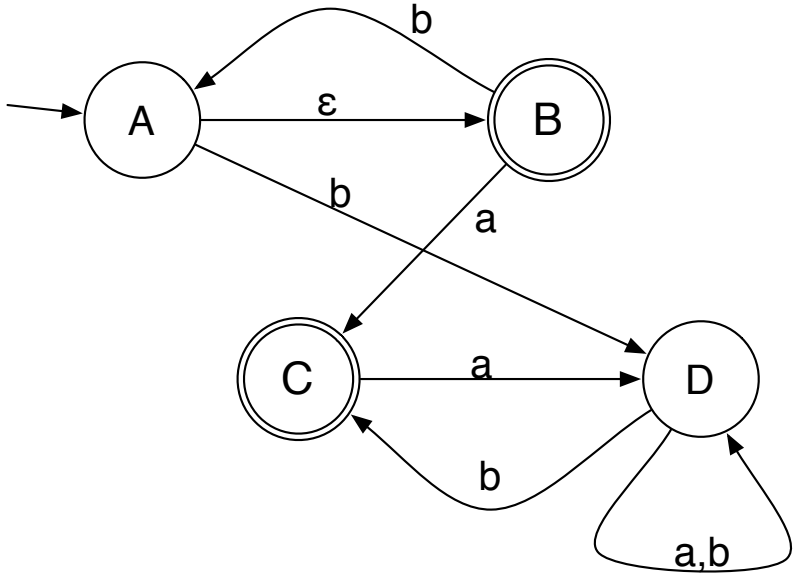
Our First Class of Machines: State Machines

Mathematically, a **state machine** consists of:

1. an alphabet Σ
2. a collection of states K
3. a transition relation $\rightarrow \quad \subseteq \quad K \times (\Sigma \cup \{\varepsilon\}) \times K$
   (where $q \xrightarrow{\sigma} q'$ means that (q, σ, q') is in the relation)
4. one initial/starting state $s \in K$.
5. a set of final/accepting states $A \subseteq K$.

finite state machine:
K is finite

deterministic state machine:
a transition *function* $\delta : K \times \Sigma \rightarrow K$.



Machine Behavior

- ✓ The machine starts in state q_0.
- ✓ It can change from state q to state q' on input σ provided that $q \xrightarrow{\sigma} q'$.
- ✓ It can change from state q to state q' spontaneously provided that $q \xrightarrow{\varepsilon} q'$.
- ✓ The machine accepts a string $w \in \Sigma^*$ if there is *at least one* path spelling out w, that starts at q_0 and ends at a state $\in F$

Derivatives

Given a language L, define

$$\partial_w L := \{ y \in \Sigma^* \mid wy \in L \}$$

That is,

$$\forall y \in \Sigma^*. \quad y \in \partial_w L \longleftrightarrow wy \in L$$

These sets are called "derivatives" of the language.

Note that

$$\partial_{ab} L = \partial_b(\partial_a L)$$

Intrinsic States

The derivatives of a language L are less commonly called its "intrinsic states," because they encode the smallest deterministic state machine for the language L.

Key idea: given what I've seen already, what further input string would make me happy?

- ✓ States: the derivative sets
- ✓ Start state: L
- ✓ Accept states: any derivative containing ε ("I'm already happy")
- ✓ Transitions:

$$S \xrightarrow{a} \partial_a S$$

Exercise

Use intrinsic states to construct the minimal *deterministic* machines for:

- ✓ $L = \{ bbc, bc, cccb \}$
- ✓ $L = \{ a^3 n \mid n \geqslant 0 \} = \{ \varepsilon, aaa, aaaaaa, \ldots \}$
- ✓ $L = \{ a^{2^n} \mid n \geqslant 0 \} = \{ a, aa, aaaa, aaaaaaaa, \ldots \}$

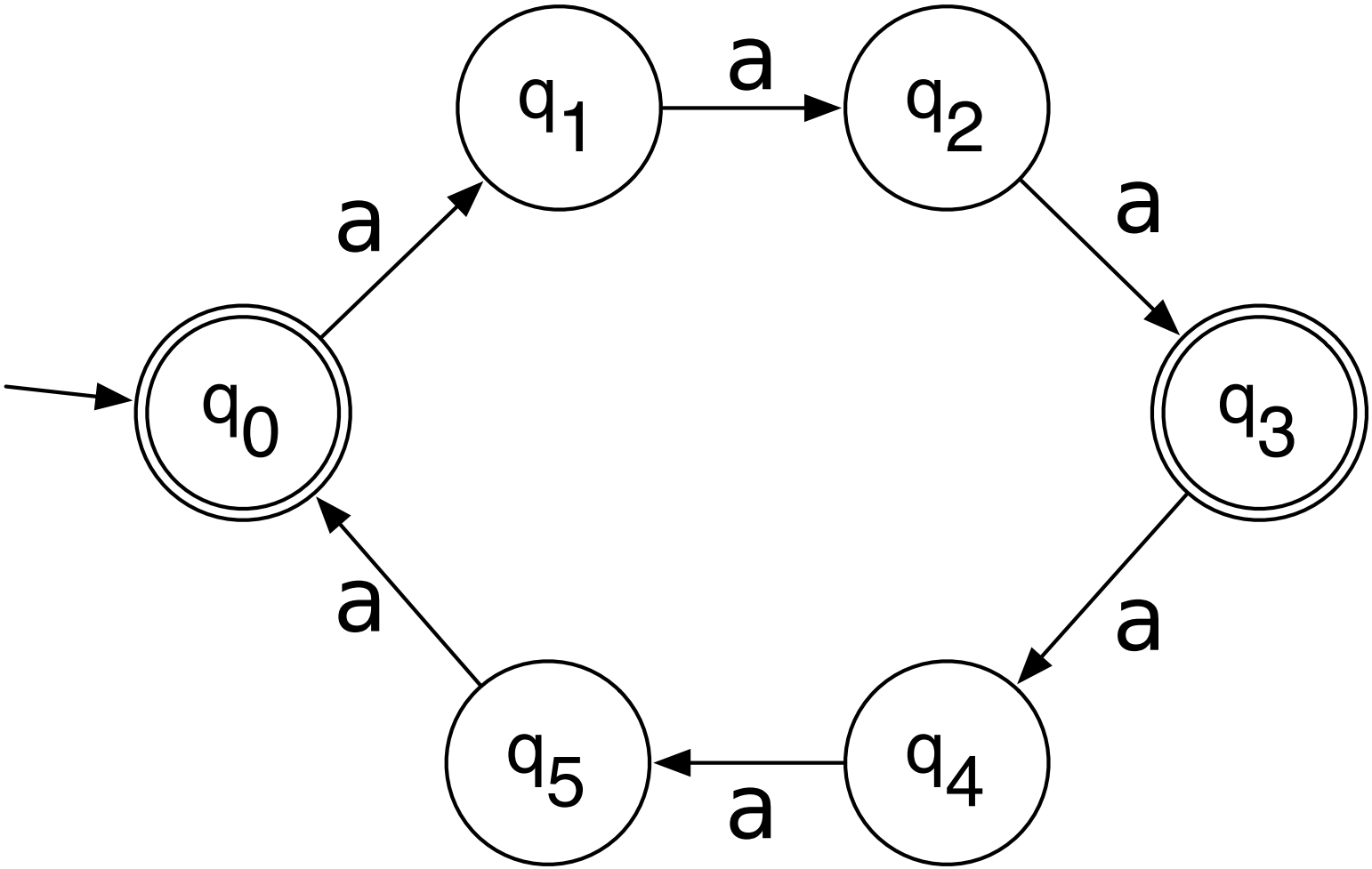
Finite State Machines

We care mostly about *finite* state machines, also known as "Finite Automata"

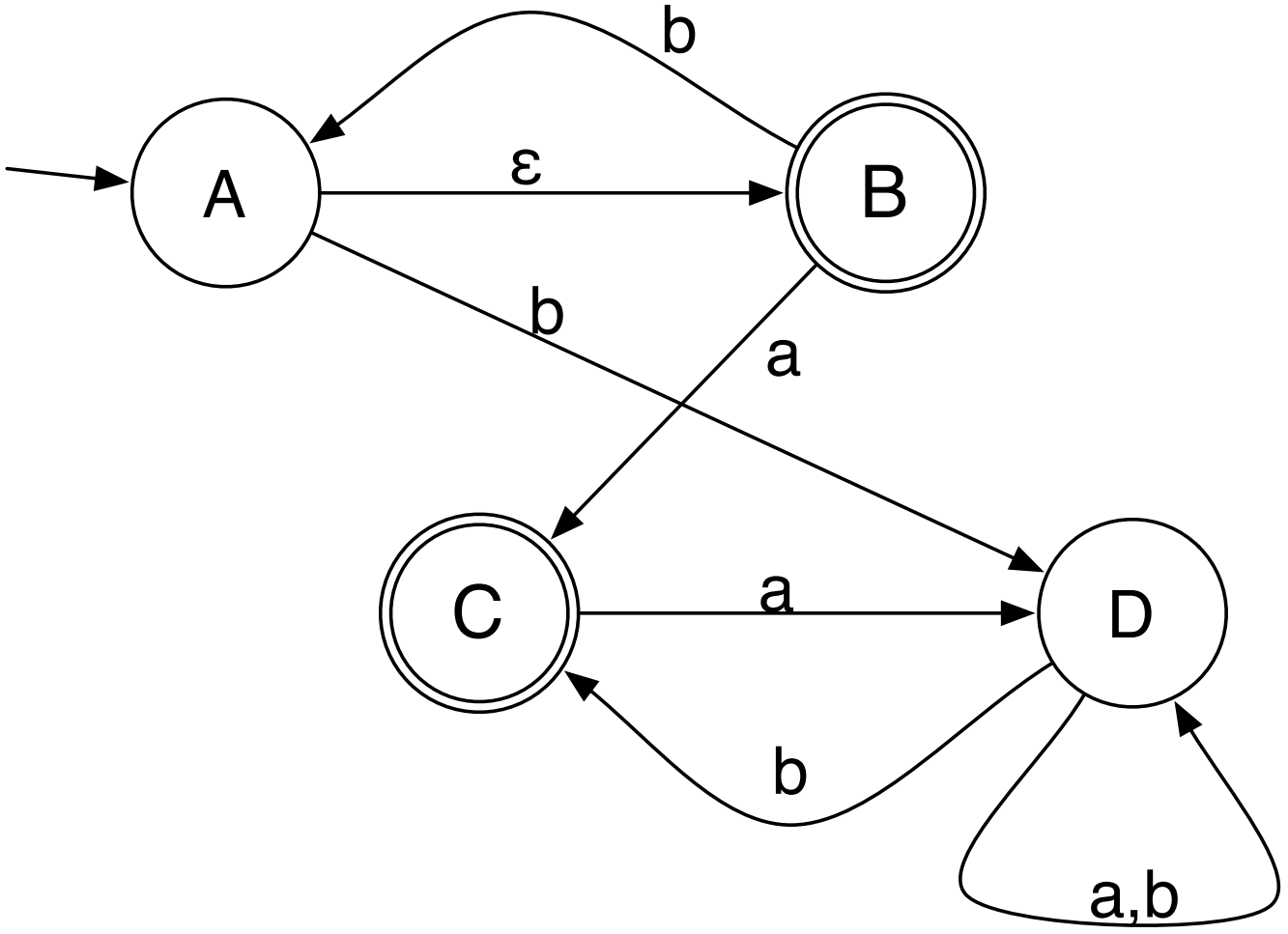
Terminology:

- ✓ DFA = Deterministic Finite Automaton = Deterministic FSM = DFSM
- ✓ NFA = Nondeterministic Finite Automaton = Nondeterministic FSM = NDFSM

What's Accepted?



What's Accepted? (bb? b? aaab? ba?)



A Jewel of Theoretical Computer Science



The following are equivalent:

- ✓ There is a DFA accepting the language L
- ✓ [Rabin and Scott] There is an NFA accepting L
- ✓ [Kleene] L is a regular set.